Model-Driven Development of Performance Sensitive Cloud Native Streaming Applications

José Ángel Bañares

Computer Science and Systems Engineering Department
COS2MOS research group, Aragón Institute of Engineering Research (I3A)
University of Zaragoza, Zaragoza, Spain
banare@unizar.es

Abstract. The number of applications that process data in a stream basis has increased significantly over recent years due to the proliferation of sensors. Additionally, in cyber-physical systems, physical and software components are deeply intertwined, adding the ability to act on the environment.
In many cases, cloud resources are used for the processing, exploiting their flexibility, but these sensor streaming applications often need to support operational and control actions that have real-time and low-latency requirements that go beyond the cost effective and flexible solutions supported by cloud platforms. The development of these applications cannot be delegated to the magical properties of frameworks and services that promise simple solutions, hiding the inherent underlying complexity of cloud resources. It raises the difficulty of developing complex streaming processing in the cloud and highlights the need for a suitable developing methodology. Moreover, during the developing life-cycle, a number of facets have to be considered such as the design of functional parallel solutions, the impact of a target cloud platform that exhibits different degrees of performance variability, or the need for more complex performance requirement support. This talk will present our experiences in developing Petri Net models for performance sensitive cloud applications thus leveraging the use of formal models in complex scenarios.

Keywords: Streaming applications · Cloud Native Applications · Performance Sensitive Applications · Petri nets.

1 Cloud Native Streaming Applications

There is an emerging interest in low latency streaming applications that consume big volumes of data. Stream processing finds application in almost every industry, business and scientific application. The sources of data can be generated from sensors, scientific instruments, simulations, social networks, business processes, etc. Data are transmitted continuously, forming a sequence of data elements known as a data stream and must be processed fast in order to control systems, take corrective/strategic actions, or react to urgent situations.

To effectively support this emerging class of applications, it is often necessary to generate workflow specifications that can be dynamically adapted – as new data becomes available in the context of the mainstream definition of Big Data as the three Vs: volume, variety and velocity [1]. With the elastic nature of many cloud environments enabling such dynamic workflow graphs to be enacted more efficiently, it might seem that data flow applications would somehow become simpler, but that is not the case. One of the challenges of data flow applications is that they must be designed with the needed level of dynamism to take account of the availability of data and the variability of the execution environment, which can be dynamically scaled out based on demand. According to [2], clouds may also be used as accelerators to improve the application time-to-completion, or to handle unexpected situations such as an unanticipated resource downtime, inadequate allocations or unanticipated queuing delays. To achieve such system, developers must meet several challenges that go beyond pure functionality. A trial-and-error implementation that tries different deployments of a streaming application over different distributed computing platforms is far from a trustworthy solution.

A cloud native application (CNA) is designed specifically to take full advantage from the cloud computing characteristics. Fehling et al. have identified the main characteristics of a cloud computing architecture in [3]: 1) The **decomposition** of the functionality in chunks of distributed functionality in such a way that that each component that made up the application can be scaled out independently. 2) The design space or operation model comprises the analysis of the application **workloads**, and the identification of how the application handles **state** and the cost of sharing information. 3) Applications are desgined taking into account **resilience** and **elasticity**: An important aspect related to the workload is to consider the dynamic nature of the cloud, which can be caused by performance variation of machines, services competiting for shared resources, and changing user quality of services requirements [4, 5].

The need for more complex performance sensitive applications callenges current development practices. The complexity of developing cost-effective and performance sensitive cloud native streaming applications has been addressed from different apporaches: 1) Cloud based frameworks that lift the level of abstraction reducing complexity and hiding resource management. 2) Collecting high-quality solutions, which are presented as patterns, to recurring problems in parallel applications and cloud platforms. 3) Developing ad-hoc performance models to predict the behavior of particular patterns on specific platforms. As far as this author knows, there is a lack of complete methodological approaches to conduct developers through the entire process of developing streaming applications as CNA, beyond the partial solutions provided by these approaches. The main shortcoming of current approaches is the non-use of formal models for helping developers to reasoning and automatize the proces of analyisis of functional and non-functional requirements. This paper covers the main characteristics of a cloud native streaming applications making reference to our works using Petri Net models for performance sensitive cloud applications.

First the requirements and a synoptical view of the proposed methodology is presented in section 2. Section 3 briefly presents a specification language to support the methodology. The behaviour of component specifications is defined by Petri Nets to support the analysis and simulation of models. Due to the large size and complexity of these systems, it is not possible to develop detailed models. We will illustrate how to use models for understanding complex behaviors, and how to calculate performance boundaries and conduct simulations in section 4. Finally, in section 5, PN models of different mechanisms to support *elasticity* and *resilience* are presented. This models allow developer to validate the mechanisms in different simulated scenarios.

2 Methodology for Performance Sensitive Streaming Applications on the Cloud

In [6–9] we have identified and presented the principles of the methodology to cope with the inherent complexity of streaming applications on the cloud. The methodology considers all involved elements at different abstraction levels.

We have identified the following *modelling requirements* that go beyond pure functionality:

- A **development process**, which is guided by the identified abstraction levels, that provides a number of modelling artifacts, analytical methods, and guidelines to support it. The methodology must address functional and non-functional requirements together with the specification of the execution infrastructure and the involved resources.
- A **specification language** to describe a streaming application as a collection of platform-independent building blocks. The language must support complementary point of views: behavioral specification of concurrent processes, transformations operated over the data flow, and structural description of components that configure the application [10–14].
- A **formal component-based** development to build models from existing components and capability to reason about the resulting composition. Reuse of components allows developers to use knowledge of their properties to predict the new system properties. Components models must provide a rich specification to facilitate the use of different analysis and prediction techniques that simplify a system design while increasing trust in its correct implementation [15].
- A **guide** of the possibilities of model reasoning for efficient and reliable design and / or optimization, combining simulation, and approximate analysis. The specification must be executable to support both *analytical analysis* and *simulation* in a synergic way.

A description of the global functionality by means of an algorithm says nothing about the structure or the components that make up the system. There are many ways in which a system can be built to provide the same functionality with different concurrent behaviours and different deployments over distributed

infrastructures [11]. Our approach to identify the elements to compound our hybrid specification can be summarised with the equation 'Specification of CDFA = Functional Entities + Communication/Synchronisation mechanisms + Data Dependencies + Resources'. The identification and characterisation of each building block of the proposed equation constitutes the basic specification element.

A *synoptical view* of our methodology include the following steps:

- **Functional Level.** The process starts by identifying the functional requirements of the problem domain and the outcome of this step is a functional model.
- **Qualitative Analysis.**The functional model analysis is used to gain identify problems and help guide the redesign of the functional model aiming to achieve the maximum level of concurrency.
- **Operational Level.** The operational level takes into account the execution platforms, and it explores the design space to select the design pattern that most effectively defines how to map processes to resources. The outcome is an operational model.
- **Quantitative Analysis.** The integration of the functional and the operational model allows the designer to evaluate performance and reward functions. The analysis can help guide the redesign of the functional and operational models to meet non-functional requirements.
- **Implementation Level.** This stage transforms the model into a flat model of processes that are deployed in a topology of cloud resources.
- **Monitoring.** The last step collects monitoring data from all used resources and applications. Collected data and developed models can help identify performance anomalies, and provide support to the autonomic principles of a Platform as a Service. The primary aim is to reduce human intervention, cost, and the perceived complexity by enabling the autonomic platform to self-manage applications [16].

The semantics of the component-based language is defined formally in terms of ordinary PNs [17] in order to translate to the methodology all the advantages derived from a mathematically based model –e.g. Analysis, Verification, or Equivalence Relations. The consideration of PNs is based on the natural descriptive power of concurrency, but also on the availability of analytic tools coming from the domain of Mathematical Programming and Graph Theory. Moreover, taking into account that PNs are executable specifications, PN models can also be simulated.

3 Data flow language specification

In [9], we presented a component based specification LANGuage of Layers and tIERS (*Langliers*) to support the methodology for building trustworthy continuous data flow applications. *Langliers* allows developers to specify the functional model as layers, the logical groupings of the functionality and components; and the operational model as tiers, the physical distribution of the functionality

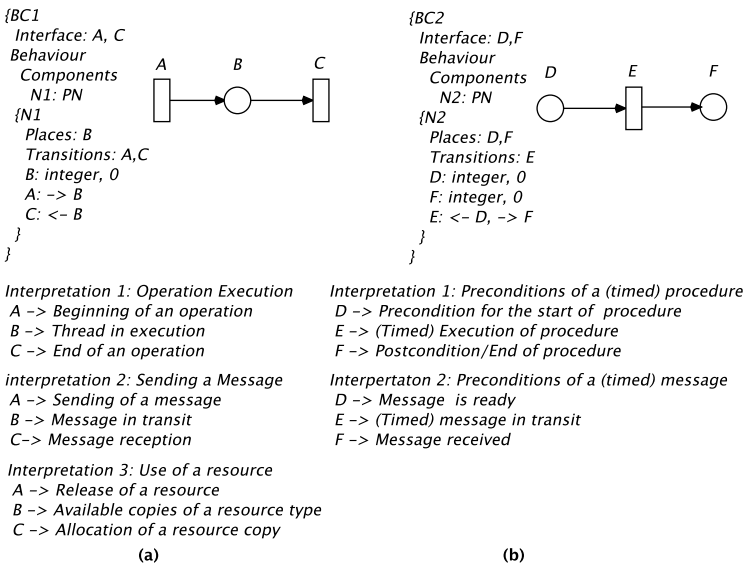

Fig. 1. Component specification of basic component primitives and interpretations.

and components on separated servers, computers, networks or remote locations. Component behaviour is defined by Petri Nets.

The constructive elements of a data stream application begins with the definition of the most basic building blocks and their interpretation as constructive primitives of distributed applications, and continues on their composition by means of simple operators, which provides the way to configure components with complex behaviours. *Langliers* represents the basic components identified in [7], **Computational** and **Data Transmission Processes**, to describe an event processing network as a graph showing various connected processing components that operate over data stream. This processing network model is an abstraction that describes the **functional** behaviour of an streaming application made up of a number of platform-independent components. The explicit network specification allows developers to visualize the functional model and apply analysis techniques. The graphical representation of a large or complex network can be somewhat unwieldy, but an hierarchical approach where components can be defined by the composition of subcomponents can simplify the specification of large models. The last remaining step to specify a complete model is to set the implementation of the functional model with the specification of the resources that will be used to execute the model. This **operational** specification can be used to conduct performance optimizations selecting a good mapping of Computational Processes and Data Transmission Processes to computational and network resources.

In *Langliers* a component is expressed in the form of an *interface* and a *behaviour* description. The **interface** specifies the services the component pro-

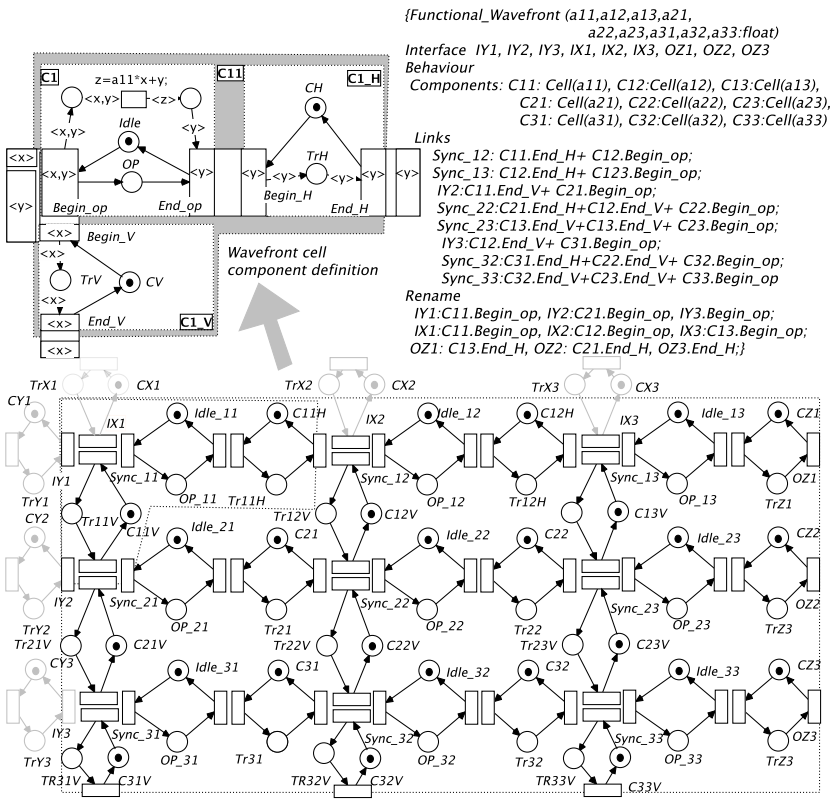

Fig. 2. Functional 3 × 3 wavefront array.

vides, and publishes its input and output **ports**. It gives information at the syntax level that enables *data type checking*, and publishes the *events* that trigger computations and state changes. This way, our data stream model follows a data driven execution model where events lead to computations that may generate events on other components. The ***behaviour*** represents components internal states and state changes. A component behavior description is specified either by one explicit PN or by the PN resulting from the behavior composition of subcomponents. It is inspired in digital systems VHDL language specification whose components specifications consist of a port declaration in the interface enumerating the events that change the component state, and the *architecture* declaration that describes either the entity's behavior or its structure. Samples of the proposed language are shown in figure 1 that shows the definition of basic component primitives and their interpretations, and figure 2 that shows the functional model of a 3x3 Wavefront array which is built with simple computational and data transmission processes.

4 Modeling for Understanding versus Modeling for Forecasting

The proposed model-driven methodology aims at providing different analysis and prediction techniques that allow developers to assess functional and non-functional properties by means of **qualitative** and **quantitative** analysis. Qualitative analysis aims to detect qualitative properties of concurrent and distributed systems, that is, to decide whether the model is correct and meets the given qualitative functional properties (e.g deadlock freedom). Qualitative PN analysis can be conducted by means of different techniques: (i) The construction of the state space of the model (reachability analysis) providing a complete knowledge of all its properties – in case state explosion does not hamper the use of this technique; (ii) Structural techniques in order to reason about some properties of the model, from the structure of the net.

In order to illustrate our methodology, we are making use of the Matrix-Vector Multiplication problem in streaming fashion, in particular, the *Wavefront Algorithm*, which represents a simple solution for large arrays [12]. The adequacy of the wavefront use case is that it is easy to formulate and present, but it shows a very complex behavior. Moreover, the literature shows a continuous effort of the research community to develop different performance models for the warfront algorithm executed on different platforms. The wavefront model is a strongly connected marked graph (a subclass of Petri nets in which each place has only one input and one output transition, being strongly connected in the sense of graph theory) [18]. The use of qualitative analysis can help to understand the behavior of the wavefront algorithm, and design solutions to obtain the maximum level of concurrency.

For quantitative analysis is importan to consider the dynamic nature of the cloud, which can be caused by performance variation of machine instances offering the same capability, and by services that are deployed, updated and destroyed all the time giving rise to a dynamic competition for shared resources [4, 5]. Due to the large size and complexity of these systems, it is not possible to develop a detailed model that captures all involved aspects. The usual approach to afford complexity in formal models such as queuing systems, is to model the essential aspects related with the behavior to be analysed, and to incorporate to the model the prediction of observable quantities by probabilistic models [19]. However, formal-model based analysis tools are only useful under certain assumptions that are not satisfied by cloud or container centres. Kazhei et al. point out three reasons that hamper the use of the tools of queuing theory: 1) The system size that involves a large number of nodes comparing to the number of nodes considered in traditional queuing analysis. 2) Involved service times must be modeled by a general probability distribution instead of a exponential distribution, which is a more convenient mathematical model. 3) The dynamic nature of these environments with dynamic loadworks and heterogeneous resources [20]. In these cases, we can use approximate methods to compute performance bounds, and complement these approximate analytical tools with simulations [20].

Streaming applications on the cloud are complex systems where customers and resources have not identical characteristics , and exponential distribution does not adequately model observed inter-arrival and service times. Therefore, traditional queuing systems are not feasible as forecasting models to obtain accurate performance evaluations. In a first approach, we labelled our model with all times following an exponential distribution. The net becomes a Stochastic Petri Net (SPN) and we translated it to the GreatSPN2.0.2 tool. The result obtained by the **GSPN analysis** showed a poor throughput. We can find the explanation of these poor results to the concurrency limitations found in the structural analysis. All computational resources will have the same throughput due to the high coupling imposed by synchronization constratins, and as a result throughput is determined by the slower resource. Intuitively, the use of a negative exponential probability distribution function which has a high coeffcient of variation with value 1, makes more likely a slower resource than the expected media with a larger number of resources.

In [18], authors present **upper** and **lower bounds** on the steady-state performance of marked graphs that can be computed effciently. In addition to the mean service time and mean inter-arrival time, the **coefficient of variation (CoV)** of resources and inter-arrival time has been proposed to introduce the dynamic nature of cloud applications and streaming applications on the cloud [4, 21, 20, 14]. Once these performance boundaries are defined, we can conduct **simulations** to explore peformance in function of the CoV of injection and processing rates. Hamzeh Khazaei et al. [20] proposes a CoV of delivery service on cloud centers with values between 0.5 and 1.4. According to these authors these values give reasonable insight into the behavior and dimensioning of cloud centers. **Simulations** showed that a CoV ranging from 0 to 1.5 covers all space of performance values between the calculated performance boundaries. With deterministic times we have performance near the upper bound, and with CoV of 1.5 performance is near the lower bound.

Previous analysis is adequated for analyse a concrete application or parallel pattern. However, to develop a complete model of each application that can be executed to obtain an operational and performance model of a cluster would be impractical. Profiling data is essential to feed models with time distribution annotations to forecast performance. For this purpose, it is necessary to provide a characterisation of different kind of applications and their effects on the remaining applications executed over the same resources. **Application profiling** is strongly associated with the workload analysis. Profiling must collect a large amount of data generated by the cloud resources and forecasting models are fed with these data to analyse resource contention and service degradation. A survey on forecasting and profiling models for cloud applications can be found in [22]. In[23, 24] we analyse performance of the Kubernetes system and develop a Reference net-based model of resource management within this **container management system**. Our model is characterised using real data from a Kubernetes deployment, and can be used as a basis to design scaleable applications that make use of Kubernetes.

5 Elasticity and Resilience

Finally, the two last properties to be considered in the development of CNA are **Component refinement** and **Management components** to support *elasticity* and *resilience*. Solutions for the latter are presented by patterns such as load balancers, or elastic queues. In [25, 26, 16, 27, 23] are presented specifications of strategies on cloud for resource management following autonomic principles at the application level for streaming and scientific workflows.

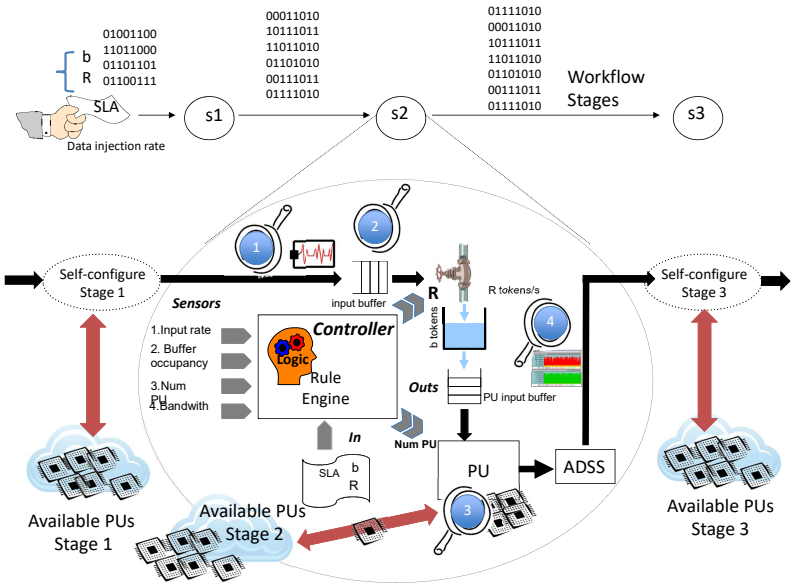

Fig. 3. System architecture and control loop for decision making in a processing node.

Formal models allow the rapid prototyping and simulation of different scenarios of complex mechanisms to support elasticy and resisilience. In [28, 29] we proposed a streaming workflow model of computation and an exception-handling mechanism for modifying at run-time the structure of a workflow. The need to close the control loop to take decisions and act on time requires a new model for analyzing and acting on IoT data that combines the cloud processing with *edge computing* or *Fog computing* [30], and **autonomic computing** techniques. It supposes the analysis of the most time-sensitive close to where data is generated and send selected data to the cloud. Additionaly, data elements are streamed from their source to their sink, and may be processed en-route (referred to **in transit** processing) [26]. This integration may imply that the associated runtime resource allocation is dependent on environmental conditions and can change for different enactments of the same workflow. In our proposal, our workflow specifications are independent of the constraints imposed by the resource allocation.

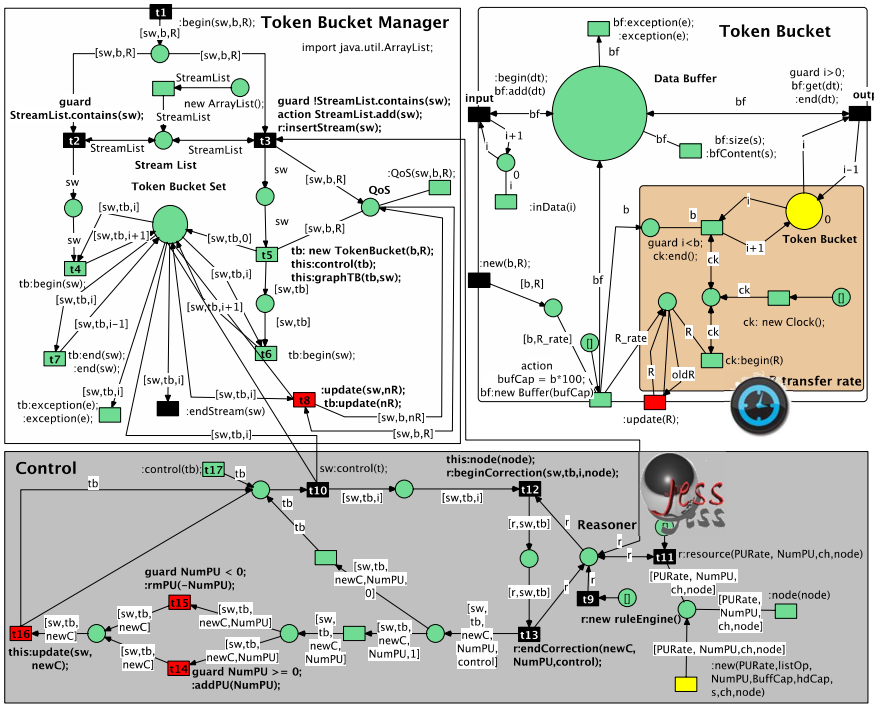

Fig. 4. Reference net model of the System architecture and control loop.

Finally, in [31] we proposed a profit-based resource management strategy for bursty data streams on shared Clouds. Even dynamic provisioning of resources may not be useful since the delay incurred might be too high – it may take several seconds to add new resources (e.g. instantiate new Virtual Machines (VMs)), and a scaling-up action might generate substantial penalties and overheads. We presented in this work an architecture and mechanisms based on the **token bucket** for the management of shared computational resources, in order to support QoS levels of several concurrent data streams and to maximize revenue of cloud providers. Figure 3 shows the system architecture and control loop for decision making in each processing node of a chain of distributed resources, and figure 4 the executable Reference net model specification implemented in Renew. Different scenarios were validated in terms of simulation. The fact that our Reference net models are executable, as they can be interpreted by Renew, allows us to use the same model to interface directly with OpenNebula from the nets: create and switch on and off real Virtual Machines (VMs), transmit data to the data centre and collect back the results. Our main contributions consists of data admission and control policies to regulate data access and manage the impact of data bursts, and a policy for resource redistribution that tries to minimize the cost of QoS penalty violation, maximising the overall profit.

Acknowledgments. This work was co-financed by the Industry and Innovation department of the Aragonese Government and European Social Funds (COS2MOS research group, ref. T93); and by the Spanish Ministry of Economy under the program "Programa de I+D+i Estatal de Investigación, Desarrollo e innovación Orientada a los Retos de la Sociedad", project id TIN2013-40809-R. I want to acknowledge members of the **COS2MOS (Computer Science for Complex System modelling)** research group that have been involved on referenced works: **Unai Arronategui**, **José Manuel Colom**, **Victor Medel**, **Rafael Tolosana**, **Omer F. Rana** (University of Cardiff) and **Congduc Pham** (Université de Pau et des Pays de l'Adour).

References

1. Laney, D.: 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group (February 2001)
2. Kim, H., Khamra, Y.E., Jha, S., Parashar, M.: An autonomic approach to integrated HPC grid and cloud usage. In: Fifth International Conference on e-Science, E-SCIENCE 2009, Oxford, UK, IEEE Computer Society (December 2009) 366–373
3. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications. Springer (2014)
4. Yeo, S., Lee, H.H.: Using mathematical modeling in provisioning a heterogeneous cloud computing environment. Computer **44**(8) (2011) 55–62
5. O'Loughlin, J., Gillam, L.: Performance evaluation for cost-efficient public infrastructure cloud use. In: Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON'14 , Cardiff, UK, September 16-18, 2014. Volume 8914 of LNCS. (2014) 133–145
6. Bañares, J.Á., Tolosana-Calasanz, R., Tricas, F., Arronategui, U., Celaya, J., Colom, J.M.: Construction of data streams applications from functional, non-functional and resource requirements for electric vehicle aggregators. The COSMOS vision. In: International Workshop on Petri Nets and Software Engineering, PNSE 2014. Number 1160 in CEUR Workshop Proceedings, Aachen, CEUR-WS.org (2014) 333–334 http://ceur-ws.org/Vol-1160/.
7. Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: Towards Petri net-based economical analysis for streaming applications executed over cloud infrastructures. In: Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON'14, Cardiff, UK, September 16-18, 2014. Volume 8914 of LNCS. (2014) 189–205
8. Tolosana-Calasanz, R., Bañares, J.Á., Rana, O., Pham, C., Xydas, E., Marmaras, C., Papadopoulos, P., Cipcigan, L.: Enforcing quality of service on opennebula-based shared clouds. In: CCGRID'14 Workshop on Data-intensive Process Management in Large-Scale Sensor Systems, DPMSS'14, Chicago, IL, USA, May 26-29, 2014, IEEE (2014) 651–659
9. Merino, A., Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: A specification language for performance and economical analysis of short term data intensive energy management services. In: Economics of Grids, Clouds, Systems, and Services - 12th International Conference, GECON'15, Cluj-Napoca, Romania, September 15-17, 2015. Volume 9512 of LNCS. (2015) 1–17

10. Etzion, O., Niblett, P.: Event Processing in Action. 1st edn. Manning Publications Co., Greenwich, CT, USA (2010)
11. Pautasso, C., Alonso, G.: Parallel computing patterns for Grid workflows. In: Proceedings of the HPDC 2006 Workshop on Workflows in Support of Large-Scale Science, WORKS 2006, June 19-23, Paris, France. (2006) 1–10
12. Yu, L., Moretti, C., Thrasher, A., Emrich, S.J., Judd, K., Thain, D.: Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions. Cluster Computing **13**(3) (2010) 243–256
13. Simmhan, Y., Kumbhare, A.G.: Floe: A continuous dataflow framework for dynamic cloud applications. CoRR **abs/1406.5977** (2014)
14. Lohrmann, B., Janacik, P., Kao, O.: Elastic stream processing with latency guarantees. In: Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on. (June 2015) 399–410
15. Seceleanu, C.C., Crnkovic, I.: Component models for reasoning. IEEE Computer **46**(11) (2013) 40–47
16. Tolosana-Calasanz, R., Bañares, J.Á., Colom, J.M.: On autonomic platform-as-a-service: Characterisation and conceptual model. In Jezic, G., Howlett, R.J., Jain, L.C., eds.: Agent and Multi-Agent Systems: Technologies and Applications. Volume 38 of Smart Innovation, Systems and Technologies. Springer International Publishing (2015) 217–226
17. Murata, T.: Petri nets: Properties, analysis and applications. In: Proceedings of IEEE. Volume 77. (April 1989) 541–580
18. Campos, J., Chiola, G., Colom, J.M., Silva, M.: Properties and performance bounds for timed marked graphs. Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on **39**(5) (1992) 386–401
19. Harrison, P.G., Patel, N.M.: Performance Modelling of Communication Networks and Computer Architectures. Addison-Wesley Longman Publishing Co., Inc. (1992)
20. Khazaei, H., Misic, J., Misic, V.: Performance analysis of cloud computing centers using m/g/m/m+r queuing systems. IEEE Transactions on Parallel and Distributed Systems **23**(5) (2012) 936–943
21. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: Observing, analyzing, and reducing variance. Proc. VLDB Endow. **3**(1-2) (September 2010) 460–471
22. Weingärtner, R., Bräscher, G.B., Westphall, C.B.: Cloud resource management: A survey on forecasting and profiling models. J. Network and Computer Applications **47** (2015) 99–106
23. Medel, V., Rana, O., Bañares, J.a., Arronategui, U.: Modelling performance and resource management in kubernetes. In: Proceedings of the 9th International Conference on Utility and Cloud Computing. UCC '16, New York, NY, USA, ACM (2016) 257–262
24. Medel, V., Rana, O., Bañares, J.Á., Arronategui, U.: Adaptive application scheduling under interference in kubernetes. In: Proceedings of the 9th International Conference on Utility and Cloud Computing, ACM (2016) 426–427
25. Tolosana-Calasanz, R., BañAres, J.Á., Pham, C., Rana, O.F.: Enforcing qos in scientific workflow systems enacted over cloud infrastructures. Journal of Computer and System Sciences **78**(5) (2012) 1300–1315
26. Tolosana-Calasanz, R., Bañares, J.A., Rana, O.F.: Autonomic streaming pipeline for scientific workflows. Concurrency and Computation: Practice and Experience **23**(16) (2011) 1868–1892

27. Tolosana-Calasanz, R., Bañares, J.Á., Pham, C., Rana, O.F.: Resource management for bursty streams on multi-tenancy cloud environments. Future Generation Computer Systems **55** (2016) 444–459
28. Tolosana-Calasanz, R., Rana, O.F., Bañares, J.A.: Automating performance analysis from Taverna workflows. In: Component-Based Software Engineering: 11th International Symposium, CBSE 2008, Karlsruhe, Germany, October 14-17, 2008. Proceedings. (2008) 1–15
29. Tolosana-Calasanz, R., Bañares, J.A., Rana, O.F., Álvarez, P., Ezpeleta, J., Hoheisel, A.: Adaptive exception handling for scientific workflows. Concurr. Comput. : Pract. Exper. **22**(5) (April 2010) 617–642
30. Yi, S., Li, C., Li, Q.: A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, ACM (2015) 37–42
31. Tolosana-Calasanz, R., Bañares, J.Á., Pham, C., Rana, O.F.: Resource management for bursty streams on multi-tenancy cloud environments. Future Generation Computer Systems **55** (2016) 444 – 459